
بسم الله الرحمن الرحيم

گزارش پروژه نهایی درس شبکه های تلفن همراه

گزارش

زهرا دهقان

اسماء حمید

فاطمه شرح دهی مقدم

آخرین ویرایش: ۳۰ مرداد ۱۴۰۴ در ساعت ۱۹ و ۱۷ دقیقه

فهرست مطالب

۱ مقدمه

۲ Server-side Component

۳ Polaris Client

۱.۳ Web Application

۲.۳ Android Mobile Client

۱.۲.۳ utils

پوشه utils یکی از پایه‌های اصلی معماری تمیز و ماژولار در این پروژه است. این فولدر شامل مجموعه‌ای از کلاس‌ها و ابزارهای کمکی است که وظایف عمومی و پرکاربرد را از منطق اصلی اپلیکیشن جدا می‌کنند. با متمرکز کردن این وظایف در یک مکان مشخص، از تکرار کد جلوگیری می‌شود و خوانایی، قابلیت نگهداری و توسعه‌ی پروژه بهبود می‌یابد.

- **مدیریت مجوزهای دسترسی**

این کلاس وظیفه مدیریت و بررسی مجوزهای ضروری اپلیکیشن را بر عهده دارد. با استفاده از این کلاس، قبل از انجام هر عملیاتی که نیاز به مجوز دارد (مانند دسترسی به مکان یا ارسال پیامک)، می‌توان از وجود مجوز اطمینان حاصل کرد.

کد ۱.۳: PermissionsUtils.kt

```
1
2  object PermissionsUtils {
3          val REQUIRED_PERMISSIONS = arrayOf(
4          Manifest.permission.ACCESS_COARSE_LOCATION,
5          Manifest.permission.SEND_SMS,
6          Manifest.permission.RECEIVE_SMS,
7          Manifest.permission.ACCESS_FINE_LOCATION,
8          Manifest.permission.READ_PHONE_STATE
9          )
10
11
12         fun hasAllPermissions(context: Context): Boolean {
13                 return REQUIRED_PERMISSIONS.all {
```

```
                        ActivityCompat.checkSelfPermission(context, it) ==
                            PackageManager.PERMISSION_GRANTED
                }
        }
}
```

- **ابزار مکان‌یابی**

این کلاس یک واسط ساده برای دریافت آخرین موقعیت مکانی کاربر فراهم می‌کند. ابتدا وجود مجوزهای لازم از `PermissionsUtils` بررسی می‌شود و سپس با استفاده از سرویس‌های Google Play، مکان کاربر به صورت ناهمگام دریافت شده و نتیجه از طریق یک callback برگردانده می‌شود.

کد ۲.۳: LocationUtils.kt

```
object LocationUtils {
        @SuppressLint("MissingPermission")
        fun getCurrentLocation(context: Context, onLocationReceived: (
            Location?) -> Unit) {
                if (!PermissionsUtils.hasAllPermissions(context)) {
                        onLocationReceived(null)
                        return
                }
                val fusedLocationClient = LocationServices.
                    getFusedLocationProviderClient(context)
                fusedLocationClient.lastLocation
                .addOnSuccessListener { location: Location? ->
                        onLocationReceived(location)
                }
                .addOnFailureListener {
                        onLocationReceived(null)
                }
        }
}
```

- **ابزار اطلاعات شبکه**

کلاس `NetworkUtils` یک ابزار تخصصی برای جمع‌آوری اطلاعات فنی شبکه تلفن همراه است. این کلاس از قابلیت‌های سیستم اندروید برای شناسایی و تحلیل شبکه‌های سلولی اطراف استفاده می‌کند.

- **متد getNetworkTypeName**

این متد نوع شبکه فعلی دستگاه را به یک رشته قابل فهم تبدیل می‌کند. با استفاده از عبارت `when`، مقادیر عددی `TelephonyManager.networkType` به نام‌های متناظر شبکه (مثل "LTE"، "5G"، "HSPA+" و غیره) ترجمه می‌شوند.

کد ۳.۳: متد getNetworkTypeName

```
fun getNetworkTypeName(manager: TelephonyManager): String {
        return when (manager.networkType) {
                TelephonyManager.NETWORK_TYPE_LTE -> "LTE"
                TelephonyManager.NETWORK_TYPE_NR -> "5G"
                TelephonyManager.NETWORK_TYPE_HSPAP -> "HSPA+"
                TelephonyManager.NETWORK_TYPE_HSPA -> "HSPA"
                TelephonyManager.NETWORK_TYPE_UMTS -> "UMTS"
                TelephonyManager.NETWORK_TYPE_EDGE -> "EDGE"
                TelephonyManager.NETWORK_TYPE_GPRS -> "GPRS"
                TelephonyManager.NETWORK_TYPE_GSM -> "GSM"
                else ->    ""
        }
}
```

- **متد getCellInfoText**

این متد وظیفه جمع‌آوری اطلاعات کامل از سلول‌های شبکه اطراف را بر عهده دارد:

* دریافت اطلاعات با telephonyManager.allCellInfo.
* پردازش داده‌ها برای هر نوع سلول، GSM: ،WCDMA ،LTE. NR
* استخراج جزئیات فنی مانند Cell ID، PLMN ID، قدرت و کیفیت سیگنال، TAC و ARFCN.
* سازماندهی داده‌ها در یک شیء JSONObject برای ذخیره یا ارسال به سرور.
* بازگشت خروجی به صورت یک Pair شامل پیام متنی و شیء JSONObject.

- **متدهای تبدیل فرکانس (استفاده داخلی)**

این متدها برای تبدیل مقادیر ARFCN/UARFCN/EARFCN/NRARFCN به فرکانس واقعی در مگاهرتز طراحی شده‌اند و اطلاعات دقیق‌تری درباره باند فرکانسی ارائه می‌دهند.

نکته

پارامتر های دست نیافته

عدم توانایی در دسترسی به مقادیر "RAC" و "Ec/N0" در برنامه‌های اندرویدی عمدتاً ناشی از محدودیت‌های نرم‌افزاری و سخت‌افزاری سیستم‌عامل است. این پارامترها، به‌ویژه Ec/N۰ که مربوط به کیفیت سیگنال شبکه‌های WCDMA است و RAC که برای شناسایی منطقه مسیریابی در شبکه کاربرد دارد، توسط بسیاری از گوشی‌ها به صورت مستقیم در اختیار هایAPI عمومی اندروید قرار نمی‌گیرند.به همین دلیل، در محیط توسعه استاندارد و با استفاده از ابزارهای معمولی مانند Android Studio، امکان استخراج مستقیم این مقادیر محدود و در بسیاری موارد غیرممکن است.

- **تست‌کننده تحویل پیامک**

این کلاس یک ابزار دقیق برای تست عملکرد ارسال و دریافت پیامک است. با استفاده از BroadcastReceiver، زمان تحویل پیامک اندازه‌گیری شده و تأخیر آن محاسبه می‌شود.

کد ۴.۳: SmsDeliveryTester.kt

```
class SmsDeliveryTester(
private val context: Context,
private val onResult: (Long) -> Unit
) {
        private val deliveryAction = "com.example.Havanet.SMS_DELIVERED"
        private val deliveryReceiver = object : BroadcastReceiver() {
                override fun onReceive(ctx: Context?, intent: Intent?) {
                        val sentTime = intent?.getLongExtra("sentTime", -1L
                            ) ?: return
                        val deliveryTime = System.currentTimeMillis()
                        val delay = deliveryTime - sentTime
                        Log.d("SmsDeliveryTester", "SMS delivery delay:
                            $delay ms")
                        context.unregisterReceiver(this)
                        onResult(delay)
                }
        }

        fun sendSms() {
                val sentTime = System.currentTimeMillis()

                val intent = Intent(deliveryAction).apply {
                        putExtra("sentTime", sentTime)}
                val pendingIntent = PendingIntent.getBroadcast(
                context,
                0,
                intent,
                PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.
                    FLAG_IMMUTABLE
                )
                context.registerReceiver(deliveryReceiver, IntentFilter(
                    deliveryAction))
                val smsManager = SmsManager.getDefault()
                smsManager.sendTextMessage(phoneNumber, null, message, null
                    , pendingIntent)
        }
}
```

۲.۲.۳ SharedViewModel : viewModel

کلاس `SharedViewModel` در پروژه نقشی کلیدی در مدیریت و به اشتراک‌گذاری داده‌ها بین کامپوننت‌های مختلف برنامه ایفا می‌کند. این کلاس با استفاده از `ViewModel` از کتابخانه `Jetpack Android`، داده‌ها را از چرخه حیات `Activity/Fragment` جدا می‌سازد، به طوری که داده‌ها در زمان تغییر پیکربندی (مانند چرخش صفحه) از بین نمی‌روند.

پیاده‌سازی و عملکرد

- **مدیریت URL پایه:** این کلاس از `MutableLiveData` برای ذخیره URL پایه سرور (baseURL_) استفاده می‌کند. این امر به کامپوننت‌های رابط کاربری امکان می‌دهد تا به صورت زنده تغییرات URL را مشاهده کرده و به آن واکنش نشان دهند. متد `initBaseURL` آدرس IP را به صورت محلی تعریف کرده و آن را در `SharedPreferences` ذخیره نموده و سپس در baseURL_ قرار می‌دهد.
- **ذخیره‌سازی و بازیابی داده‌ها:** هسته اصلی این `ViewModel` در متدهای `saveData`، `loadData` و `clearData`قرار دارد که برای مدیریت داده‌های کلید-مقدار (Key-Value) به کار می‌روند. این متدها از `SharedPreferences` به‌عنوان یک ابزار سبک‌وزن برای ذخیره‌سازی داده‌های ساده استفاده می‌کنند.
  - **متد `saveData`:** این متد می‌تواند انواع مختلف داده‌ها (`String`، `Boolean`، `Int`، `Float`، `Long`) را بر اساس نوع آن‌ها در `SharedPreferences` ذخیره نماید. این طراحی یک راه‌حل عمومی و انعطاف‌پذیر برای ذخیره‌سازی داده‌ها فراهم می‌کند.
  - **متد `loadData`:** این متد داده‌ها را با یک کلید مشخص و یک مقدار پیش‌فرض از `SharedPreferences` بازیابی می‌کند. با استفاده از `Generics` (`<T>`) و یک عبارت `when`، این متد قادر است داده‌ها را به نوع صحیح خود تبدیل کرده و برگرداند.
  - **متد `clearData`:** امکان حذف یک مقدار خاص از `SharedPreferences` را با استفاده از کلید آن فراهم می‌کند. این قابلیت برای مدیریت داده‌های حساس (مانند توکن‌های احراز هویت) که باید پس از خروج کاربر یا در شرایط خاص حذف شوند، بسیار حیاتی است. این متد با فراخوانی `editor.remove(key)` داده مرتبط را حذف می‌کند.
- **رویکرد معماری:** استفاده از `ViewModel` به عنوان یک مخزن مرکزی برای داده‌های اشتراکی، از وابستگی مستقیم سایر بخش‌های برنامه به `SharedPreferences` جلوگیری می‌کند. این کار باعث می‌شود که کد تمیزتر، قابل نگهداری‌تر و تست‌پذیرتر باشد، زیرا منطق ذخیره‌سازی و بازیابی داده‌ها در یک مکان واحد متمرکز شده است.

۳.۲.۳ MyForegroundService : service

MyForegroundService به عنوان یک سرویس پس‌زمینه در پروژه، نقش حیاتی در جمع‌آوری و ارسال مداوم اطلاعات فنی به سرور بک‌اند را ایفا می‌کند. این سرویس به صورت *Foreground* طراحی شده تا با نمایش یک نوتیفیکیشن دائمی به کاربر، از بسته شدن ناگهانی آن توسط سیستم‌عامل اندروید جلوگیری شود. این ویژگی برای عملیات‌های طولانی‌مدت و بدون وقفه ضروری است.

پیاده‌سازی و عملکرد

- **زمان‌بندی هوشمند:** در زمان راه‌اندازی، سرویس یک *Handler* را فعال می‌کند که هر ۱۰ ثانیه یک بار متد `collectAndSendData()` را فراخوانی می‌کند. این رویکرد، پایداری و عملکرد بهینه سرویس را تضمین می‌کند.
- **جمع‌آوری داده:** متد `collectAndSendData()` وظیفه جمع‌آوری اطلاعات را بر عهده دارد. در این مرحله، سرویس از کلاس‌های `NetworkUtils` و `LocationUtils` برای دریافت جزئیات دقیق شبکه (مانند نوع و اطلاعات سلول‌ها) و موقعیت جغرافیایی دستگاه استفاده می‌کند. این طراحی، نشان‌دهنده یک معماری ماژولار و تمیز است که وظایف را به ابزارهای تخصصی خود در پکیج `utils` واگذار می‌کند.
- **ارسال به سرور:** پس از جمع‌آوری داده‌ها و ترکیب آن‌ها در یک `JSONObject`، اطلاعات به متد `sendToBackend()` ارسال می‌شود. این ارسال به صورت *Asynchronous* و با استفاده از کتابخانه `OkHttp` انجام می‌شود تا رابط کاربری اصلی (UI) مسدود نشود.
- **مدیریت پیکربندی:** آدرس سرور و توکن احراز هویت به صورت پویا از یک `SharedViewModel` بارگذاری می‌شوند که نشان‌دهنده جداسازی داده‌های پیکربندی از منطق اصلی سرویس است.
- **پایش و اشکال‌زدایی:** در صورت موفقیت یا شکست در ارسال داده، پیام‌های مربوطه در `Logcat` ثبت می‌شوند که برای رصد و اشکال‌زدایی عملکرد سرویس بسیار مفید است.

۴.۲.۳ تحلیل اکتیویتی‌ها

۱.۴.۲.۳ MainActivity

`MainActivity` به عنوان دروازه ورود به بخش اصلی اپلیکیشن عمل می‌کند و وظایف مهمی مانند مدیریت جریان ورود کاربر، درخواست مجوزهای ضروری و راه‌اندازی سرویس‌های پس‌زمینه را بر عهده دارد.

- **مدیریت جریان کاربر:** این اکتیویتی اولین نقطه‌ای است که پس از اجرای برنامه بررسی می‌کند که آیا کاربر با موفقیت ثبت‌نام کرده است یا خیر. این کار با بررسی مقدار `isRegistered` در `SharedViewModel` انجام می‌شود. اگر کاربر قبلاً احراز هویت نشده باشد، به صورت خودکار به `RegisterActivity` هدایت می‌شود تا فرآیند ورود یا ثبت‌نام را تکمیل کند.
- **درخواست مجوزها:** در صورت احراز هویت موفق، `MainActivity` مسئولیت درخواست مجوزهای لازم (مانند دسترسی به مکان و شبکه) را به عهده می‌گیرد. این کار از طریق متد `onRequestPermissionsResult()` مدیریت می‌شود که پس از دریافت پاسخ کاربر، تصمیم می‌گیرد که برنامه به کار خود ادامه دهد یا بسته شود.
- **راه‌اندازی رابط کاربری و سرویس‌ها:** پس از تأیید مجوزها، متد `initUI()` رابط کاربری اصلی شامل نوار ابزار و منوی ناوبری پایینی را راه‌اندازی می‌کند. مهم‌تر از آن، در همین مرحله سرویس `MyForegroundService` برای شروع جمع‌آوری داده‌ها به صورت پس‌زمینه فعال می‌شود. این رویکرد جداسازی کامل رابط کاربری از منطق کسب‌وکار را نشان می‌دهد.

مسیر Layout Resource File :

« app\src\main\res\layout\activity_main.xml »

۲.۴.۲.۳ RegisterActivity

`RegisterActivity` یک واسط کاربری متمرکز برای فرآیندهای حساس ثبت‌نام و ورود به سیستم است و تمام ارتباطات لازم با سرور احراز هویت را مدیریت می‌کند.

- **واسط کاربری پویا:** این اکتیویتی دارای یک حالت دوقطبی است که به کاربر اجازه می‌دهد بین «ورود» و «ثبت‌نام» جابه‌جا شود. این تغییر حالت، عنوان صفحه و متن دکمه‌ها را به صورت پویا تغییر می‌دهد تا تجربه کاربری بهتری فراهم شود.
- **ارتباط امن با سرور:** متد `sendToBackend()` از کتابخانه OkHttp برای ارسال اطلاعات احراز هویت به صورت امن استفاده می‌کند. این متد بر اساس وضعیت فعلی (ورود یا ثبت‌نام)، درخواست را به یکی از دو نقطه پایانی `/auth/login/` یا `/auth/signup/` ارسال می‌کند.
- **مدیریت پاسخ سرور:** پس از دریافت پاسخ موفق، توکن‌های `access` و `refresh` از پاسخ JSON استخراج و با استفاده از `SharedViewModel` در `SharedPreferences` ذخیره می‌شوند. این توکن‌ها برای درخواست‌های بعدی به سرور استفاده خواهند شد. سپس برنامه کاربر را به `MainActivity` هدایت می‌کند و اکتیویتی فعلی را می‌بندد تا از دسترسی غیرمجاز به اطلاعات جلوگیری شود.
- **مدیریت خطا:** در صورت بروز خطای سمت سرور (مانند نام کاربری یا رمز عبور نامعتبر)، یک پیام خطای واضح به صورت `Toast` به کاربر نمایش داده می‌شود تا از تجربه ناموفق خود آگاه شود.

مسیر Layout Resource File :

« app\src\main\res\layout\activity_register.xml »

۳.۴.۲.۳ ProfileActivity

`ProfileActivity` به کاربر امکان مشاهده جزئیات پروفایل و کنترل جلسه خود را می‌دهد. این اکتیویتی نشان‌دهنده نحوه استفاده از توکن‌های احراز هویت برای دسترسی به اطلاعات حفاظت‌شده است. با کلیک بر علامت آدمک در خط بالای صفحه اصلی میتوان وارد این بخش شد.

- **دریافت اطلاعات پروفایل:** متد `fetchProfile()` مسئولیت دریافت اطلاعات کاربر را بر عهده دارد. این متد با استفاده از `access token` ذخیره‌شده، یک درخواست `GET` به نقطه پایانی `/auth/profile/` ارسال می‌کند. این رویکرد تضمین می‌کند که فقط کاربران احراز هویت‌شده به اطلاعات خود دسترسی داشته باشند.

- **نمایش اطلاعات:** اطلاعات دریافتی (مانند شماره تلفن و نقش کاربری) در کامپوننت‌های سفارشی `ProfileInfoItemView` نمایش داده می‌شوند. این استفاده از ویوهای سفارشی، به یکپارچگی و طراحی منسجم رابط کاربری کمک می‌کند.
- **خروج امن:** متد `performLogout()` فرآیند خروج امن را مدیریت می‌کند. این متد با ارسال `token refresh` به نقطه پایانی `/auth/logout/`، توکن‌ها را در سرور باطل می‌کند. سپس با استفاده از `SharedViewModel`، تمام توکن‌ها و وضعیت ثبت‌نام (`isRegistered`) را از حافظه محلی دستگاه حذف می‌کند.
- **قطع سرویس پس‌زمینه:** پس از خروج موفق، `MyForegroundService` متوقف می‌شود تا جمع‌آوری داده‌ها به پایان برسد. سپس کاربر به صفحه `RegisterActivity` هدایت می‌شود و فرآیند ورود مجدد آغاز می‌گردد. این کار امنیت داده‌ها و حریم خصوصی کاربر را تضمین می‌کند.

مسیر Layout Resource File :

« app\src\main\res\layout\activity_profile.xml »

۵.۲.۳ Fragments : UI

در این بخش، به جزئیات هر فرگمنت که در `MainActivity` نمایش داده می‌شود، پرداخته می‌شود.

۱.۵.۲.۳ InformationFragment

`InformationFragment` یکی از مهم‌ترین بخش‌های رابط کاربری اپلیکیشن است که وظیفه **نمایش اطلاعات لحظه‌ای** مربوط به موقعیت مکانی و جزئیات فنی شبکه تلفن همراه را بر عهده دارد. این فرگمنت با استفاده از معماری MVVM (Model-View-ViewModel) پیاده‌سازی شده و داده‌ها را به صورت مستقیم از لایه‌های `utils` و `viewmodels` دریافت می‌کند.

پیاده‌سازی و عملکرد

- **اتصال به هاViewModel:** این فرگمنت به دو ViewModel متصل می‌شود:
  - `InformationViewModel`: یک ViewModel محلی که برای مدیریت داده‌های مربوط به خود فرگمنت (مانند متن دکمه) استفاده می‌شود.
  - `SharedViewModel`: یک ViewModel مشترک که برای بررسی وضعیت احراز هویت کاربر (`isRegistered`) و دسترسی به داده‌های عمومی مانند Base URL استفاده می‌شود.
- **بروزرسانی لحظه‌ای:** برای نمایش اطلاعات لحظه‌ای، از یک `Handler` و `Runnable` استفاده شده است. متد زیر هر یک ثانیه اجرا می‌شود تا اطلاعات جدید مکان و شبکه را جمع‌آوری و روی UI نمایش دهد.

```
۱ private fun updateLocationAndNetworkInfo() {
۲         LocationUtils.getCurrentLocation(requireContext()) { location ->
۳                 if (location != null) {
```

```
۴                           latitude = location.latitude
۵                           longitude = location.longitude
۶                           binding.latitudeText.text = "
                                latitude" binding.longitudeText.text = "longitude"
۷                   } else {
۸                           binding.latitudeText.text = "-"
۹                           binding.longitudeText.text = "-"
۱۰                  }
۱۱          }
۱۲          binding.cellinfoTable.removeAllViews()
۱۳          val (_, cellJson) = NetworkUtils.getCellInfoText(requireContext())
۱۴          addCellInfoToTable(cellJson)
۱۵  }
```

- **مدیریت وضعیت کاربر**: قبل از نمایش اطلاعات، فرگمنت وضعیت `isRegistered` را بررسی می‌کند. اگر کاربر احراز هویت نشده باشد، کارت‌های نمایش اطلاعات (`locationCard` و `dataCard`) مخفی شده و یک پیام خطا نمایش داده می‌شود.
- **نمایش اطلاعات**:

- **موقعیت مکانی**: مختصات جغرافیایی (`longitude latitude,`) با استفاده از `LocationUtils.getCurrentLocation()` دریافت و نمایش داده می‌شود.
- **اطلاعات شبکه**: جزئیات فنی شبکه با استفاده از `NetworkUtils.getCellInfoText()` دریافت شده و به صورت پویا در یک TableLayout قرار می‌گیرند.

- **تعامل با نقشه‌ها**: دکمه Open in Map یک `Intent` ایجاد می‌کند تا مختصات روی نقشه OpenStreetMap (OSM) نمایش داده شود.

نمایش داده‌های پویا در جدول

- **تولید جدول پویا**: متد `addCellInfoToTable()` برای هر زوج کلید-مقدار یک سطر جدید در TableLayout ایجاد می‌کند.
- **قالب‌بندی خوانا**: متد `createTableRow()` دو `TextView` (یکی برای برچسب و دیگری برای مقدار) در یک سطر قرار می‌دهد تا اطلاعات به صورت شفاف نمایش داده شوند.

مدیریت چرخه حیات فرگمنت

- `onCreateView()`: ساخت View فرگمنت و شروع بروزرسانی‌های دوره‌ای.
- `onDestroyView()`: حذف `Runnable` از `Handler` برای جلوگیری از Memory Leak و تنظیم `_binding = null` برای کمک به Garbage Collection.

مسیر Layout Resource File :

« app\src\main\res\layout\fragment_information.xml »

۲.۵.۲.۳ SettingFragment

SettingFragment یک فرگمنت کلیدی در رابط کاربری اپلیکیشن است که به کاربر امکان تعریف آستانه‌ها و مقادیر رنگی برای نمایش کیفیت سیگنال شبکه‌های مختلف را می‌دهد. این فرگمنت با فراهم کردن یک واسط کاربری پویا، امکان **شخصی‌سازی نمایش داده‌های فنی** را به صورت بصری فراهم می‌سازد.

پیاده‌سازی و عملکرد

- **اسپینرها (Spinners):** این فرگمنت دارای سه Spinner وابسته به هم است:
  - spinnerTech: انتخاب نوع شبکه، 2G، 3G، 4G، 5G.
  - spinnerType: بر اساس انتخاب تکنولوژی، گزینه‌های مربوط به نوع سیگنال (مانند rsrp یا rsrq برای 4G) را بارگذاری می‌کند.
  - spinnerNumber: انتخاب تعداد سطوح رنگی (از ۳ تا ۵۰)، که به صورت مستقیم بر تعداد باکس‌های رنگی قابل تعریف تأثیر دارد.
- **انتخاب رنگ:** با استفاده از انتخابگر رنگ، رنگ انتخاب‌شده در متغیر currentColor ذخیره می‌شود. کاربر می‌تواند با کلیک روی دکمه مربوطه، دیالوگ showLevelInputDialog() را برای ثبت جزئیات سطح رنگ فراخوانی کند.
- **دیالوگ ورودی سطح:** متد showLevelInputDialog() یک دیالوگ سفارشی نمایش می‌دهد که شامل ورودی‌های سطح، حداقل و حداکثر است. این متد ورودی‌ها را اعتبارسنجی کرده و امکان ویرایش مقادیر سطح‌های قبلی را نیز فراهم می‌کند. این متد شروط صلاحیت هر یک از این ورودی ها را نیز به عهده دارد.
- **به‌روزرسانی رابط کاربری:** متد updateColorViews() برای هر سطح، یک ویو جدید ایجاد کرده و رنگ و سطح آن را نمایش می‌دهد. همچنین برای هر ویو یک OnClickListener تعریف می‌شود تا کاربر بتواند در صورت نیاز وارد حالت ویرایش شود.

ارتباط با سرور و منطق تجاری

- **ثبت نهایی:** دکمه **ثبت نهایی** توسط متد createAndSetupFinalizeButton() ایجاد می‌شود. این متد پیش از ارسال، داده‌های کاربر را اعتبارسنجی کرده و در قالب JSON آماده می‌کند.
- **ارسال داده‌ها:** متد Final_send() مسئول ارسال داده‌ها به سرور است. این متد با استفاده از SharedViewModel، توکن احراز هویت و Base URL را دریافت کرده و از کتابخانه OkHttp برای ارسال درخواست POST به نقطه پایانی /thresholds/create/ استفاده می‌کند.
- **بازخورد به کاربر:** در صورت موفقیت، یک پیام Toast نمایش داده شده و رابط کاربری ریست می‌شود.

مدیریت چرخه حیات فرگمنت

- onCreateView(): ایجاد رابط کاربری و انجام تنظیمات اولیه.
- onDestroyView(): آزادسازی منابع و تنظیم _binding = null برای جلوگیری از Memory Leak.

مسیر Layout Resource File :

« app\src\main\res\layout\fragment_setting.xml »

۳.۵.۲.۳ TestsFragment

TestsFragment یکی از فرگمنت‌های اصلی اپلیکیشن است که به کاربر امکان اجرای تست‌های عملکردی شبکه و سرویس‌های موبایل را می‌دهد. این فرگمنت با فراهم کردن یک واسط کاربری ساده، تست‌های مختلفی مانند پینگ، سرعت آپلود/دانلود و تأخیر پیامک را به صورت خودکار و زمان‌بندی‌شده انجام می‌دهد.

پیاده‌سازی و عملکرد

- **رابط کاربری تعاملی:** این فرگمنت دارای دکمه‌هایی برای هر تست (پینگ، وب، آپلود، دانلود، DNS و SMS) است. با کلیک روی هر دکمه، تست مربوطه آغاز می‌شود.
- **مدیریت وضعیت دکمه‌ها:** متد setAllButtonsEnabled() وضعیت فعال یا غیرفعال بودن تمام دکمه‌ها را مدیریت می‌کند. هنگام اجرای یک تست، تمام دکمه‌ها غیرفعال می‌شوند تا از اجرای همزمان چند تست جلوگیری شود.
- **بروزرسانی زنده رابط کاربری:** با استفاده از LiveData و testsViewModel، نتایج تست‌ها به صورت لحظه‌ای روی UI نمایش داده می‌شود. این طراحی باعث می‌شود که UI حتی در تست‌های طولانی نیز پاسخگو باقی بماند.

اجرای تست‌ها و منطق مربوطه

- **متد startRepeatedTest():** هسته اصلی اجرای تست‌ها است.
  - **زمان‌بندی:** با استفاده از lifecycleScope و Coroutines، تست انتخاب‌شده را به مدت ۲ دقیقه و با فاصله زمانی ۱۰ ثانیه تکرار می‌کند.
  - **همگامی با UI:** عملیات شبکه در Dispatchers.IO اجرا شده و سپس نتایج با Dispatchers.Main به UI منتقل می‌شوند تا از مسدود شدن رابط کاربری جلوگیری شود.
- **تست‌های مختلف:**
  - performPingTest(): ارسال پینگ به آدرس 8.8.8.8و محاسبه زمان پاسخ.
  - testWebResponseTime(): محاسبه زمان اتصال به https://www.google.com/search?q=google.com.
  - testUploadSpeed(): اندازه‌گیری سرعت آپلود داده به سرور تستی.
  - testDownloadSpeed(): اندازه‌گیری سرعت دانلود از سرور تستی.
  - testDnsTime(): محاسبه زمان پاسخ DNS برای www.google.com.
  - testSmsDeliveryDelay(): محاسبه زمان تأخیر ارسال/دریافت پیامک با استفاده از SmsDeliveryTester و مدیریت ناهمگام با Coroutines.

مدیریت داده و ارتباط با سرور

- **ارتباط با `SharedViewModel`:** برای دسترسی به Base URL و Access Token از `SharedViewModel` استفاده می‌شود تا مدیریت داده‌های مشترک متمرکز باشد.
- **ارسال نتایج به بک‌اند:** متد `()sendTestResultToBackend` نتایج تست‌ها را در قالب درخواست `POST` و با فرمت JSON به سرور ارسال می‌کند. احراز هویت از طریق `Token Bearer` انجام می‌شود. این کار امکان ذخیره‌سازی و تحلیل نتایج در سمت سرور را فراهم می‌کند.
- **مدیریت چرخه حیات:** در متد `()onDestroyView`، ارجاع `binding` برابر `null` قرار داده می‌شود تا از Memory Leak جلوگیری گردد.

مسیر Layout Resource File :

« app\src\main\res\layout\fragment_tests.xml »

۶.۲.۳ ProfileInfoItemView : customViews

ProfileInfoItemView یک کامپوننت UI سفارشی است که به منظور نمایش یکپارچه اطلاعات کاربر (مانند نام کاربری، رمز عبور یا نقش) در صفحه *ProfileActivity* طراحی شده است. این رویکرد به کدنویسی تمیزتر و قابل نگهداری کمک می‌کند، زیرا از تکرار کد مربوط به طراحی ویوهای مشابه جلوگیری می‌کند. همچنین به این روش، در صورت نیاز، گسترش این بخش ساده تر می شود.

۱. پیاده‌سازی و ساختار

- **کلاس:** این ویو از کلاس پایه `LinearLayout` ارث‌بری می‌کند و می‌تواند عناصر داخلی خود را به صورت خطی (افقی یا عمودی) سازماندهی کند.
- **اتصال به طرح‌بندی:** در بلوک `init`، فایل XML مربوط به طرح‌بندی (`R.layout.profile_info_item`) با استفاده از `LayoutInflater` به ویو متصل می‌شود.
- **دریافت ویژگی‌های XML:** با استفاده از `context.obtainStyledAttributes`، ویو می‌تواند ویژگی‌های سفارشی تعریف شده در فایل XML مانند `app:icon` یا `app:text` را بخواند و ظاهر ویو را از طریق XML سفارشی‌سازی کند.

۲. عملکرد و قابلیت‌های کلیدی

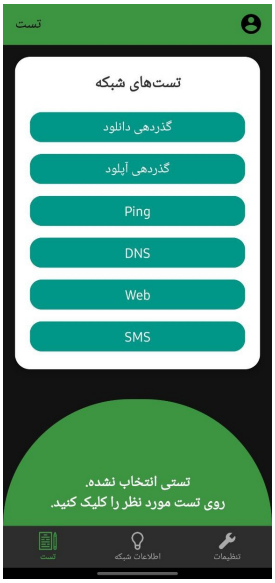
قابلیت استفاده مجدد: توسعه‌دهنده می‌تواند به جای تکرار کد، از تگ `<com.example.Havanet.customviews.ProfileInfoItemView>` در XML استفاده کند.

- **کپسوله‌سازی منطق:** منطق مربوط به نمایش آیکون و متن داخل کلاس قرار دارد و کد *ProfileActivity* تمیزتر می‌شود.
- **متدهای عمومی:**

- `setValue(value: String)` تنظیم متن ویو از طریق کد Kotlin/Java.
- `getValue()`: دریافت متن فعلی ویو برای ذخیره یا استفاده از اطلاعات.

مسیر Layout Resource File :

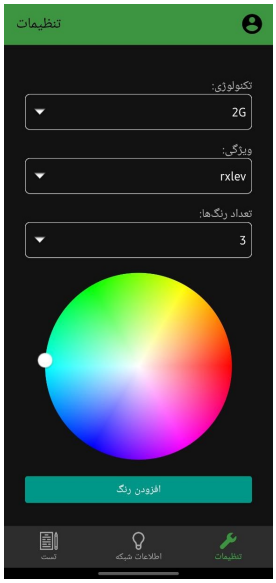
« app\src\main\res\layout\profile_info_item.xml »



(ب) تست ها

(آ) ورود/ثبت نام

(ه) حساب کاربری



(د) تنظیمات

(ج) اطلاعات

شکل ۱.۳: تصاویر صفحات نام برده شده

۴ تحلیل و نتجه گیری